Design and implementation of a platform for hyperconnected cyber physical systems

Ian Thomas, Shinji Kikuchi, Emmanuel Baccelli, Kaspar Schleiser, Joerg Doerr, Andreas Morgenstern

Design and implementation of a platform for hyperconnected cyber physical systems

Ian Thomas[c], Shinji Kikuchi[d], Emmanuel Baccelli[b], Kaspar Schleiser[b], Joerg Doerr[a], Andreas Morgenstern[a]

[a] *Fraunhofer IESE, Fraunhofer-Platz 1, 67663 Kaiserslautern, Germany*
[b] *Inria, 1 rue Honoré d'Estienne d'Orves, 91120 Palaiseau, France*
[c] *FUJITSU RunMyProcess, 3 Rue de Gramont, 75002 Paris, France*
[d] *FUJITSU LABORATORIES LTD. 1-1, Kamikodanaka 4-chrome, 211-8588 Kawasaki, Japan*

***Abstract*— The Internet of Things (IoT) is an area of growing importance as more and more computing capability becomes embedded into real world objects and environments. But at the same time IoT is just one component of a widespread shift towards a new age of federation, combining with other trends such as cloud computing, blockchain and automation to create a new *hyperconnected infrastructure*. This infrastructure will emerge from the convergence of traditional, cloud and IoT-based models of computing, creating a more decentralised, secure and democratic computing platform for the future. But while bringing significant benefits, federation also brings significant problems – in particular the complexity of building, integrating and managing systems built using highly distributed and heterogeneous platforms. In this paper we discuss our work on modelling, deployment and management for this new converged computing environment, leveraging previous work on domain languages, cloud computing and the Web of Things to accelerate and democratize the development of real world hyperconnected systems.**

I. INTRODUCTION

The Internet of Things (IoT), is an emerging computing paradigm in which a wide range of physical objects - such as machines, vehicles, appliances, wearables etc. - are equipped with sensing, processing and connectivity features. Analyst firm Gartner predicts that the number of deployed IoT devices will grow from 5 Billion in 2015 to 25 Billion in 2020 [3], making IoT one of the most important trends in the field of computer science [29].

At the same time we anticipate that IoT will combine with the cloud to create a new *hyperconnected infrastructure* - a distributed computing fabric consisting of multiple cloud platforms and billions of smart devices [16] [32]. Today, however, these environments are largely viewed as separate, with distinct technologies, tools and management approaches applied to each. Beyond the collection of device data in the cloud for further analysis there is typically very little functional overlap between the two worlds in today's pioneering deployments, limiting the utility of the whole.

But as we drive towards increasingly distributed infrastructures, today's centralized cloud-based processing models will need to evolve; real world data is simply too massive, complex and time sensitive to always send to the cloud for processing.

Being able to quickly build, test and evolve systems deployed across the full range of cloud and IoT infrastructure will therefore become critical to placing intelligence in the right place and meeting the needs of next generation digital systems. But today's IoT is not well suited to fast iteration and rapid, continual deployment, while the divergence in tools, practices and skills between the cloud and IoT is often a barrier to mainstream adoption. Furthermore current work on 'fog' computing often focuses on relatively powerful in-network infrastructure or the use of high-capability devices such as the Raspberry Pi, but these approaches cannot reach all the way to the furthest edges of the network where we expect to see huge numbers of very low cost but highly constrained devices deployed over the next few years as IoT penetrates more deeply into the fabric of society.

In this paper, we therefore outline our work to extend the rapid development, deployment and evolution characteristics of the cloud into the most constrained corners of the IoT, enabling fast and iterative development of converged *hyperconnected* solutions in line with cloud best-practices such as modelling, continuous deployment [24], micro-services and serverless execution [14]. In this way our work aims to reduce the significant technical and skills-based barriers to creating hyperconnected systems, unify the tools and practices necessary to deliver them and create a more agile, responsive and configurable IoT – encompassing every kind of node from the most powerful to the most constrained.

II. FROM A PASSIVE TO AN ACTIVE IOT

In previous work we described our vision to enable composite business ecosystems, a new kind of organizational paradigm in which people co-create value by connecting

resources [50]. As part of this work we introduced RunMyProcess [10], a platform that simplifies the creation of complex, cloud-based applications. By offering users a way to easily model, integrate and deploy processes in the cloud, RunMyProcess simplifies the deployment and management of business functionality for organizations around the world.

During this previous work we were primarily focused on extending the ability of our platform to easily connect to resources within the Web of Things [31] and on enabling the use of IoT-based data within real-time business processes as part of our vision of composition. During real-world testing [34,35], however, we discovered that collecting data was not enough; in fact we found a range of use cases in which we needed to move more of the intelligence out of the cloud and into the nodes themselves. Without this ability we found ourselves unable to deal with issues in real time and often overloaded with the costs of processing worthless data.

In trying to address these issues, however, we quickly ran into several problems due to the differing lifecycles of cloud and IoT based systems. Changes to the overall system that seemed trivial could be addressed on the cloud side in minutes but would often cause significant delays as we customized firmware on the device side. Rolling out updates would be instant for the cloud elements but be far more complex for the device side. Finding skills for the IoT aspects of the development was a challenge, with deep expertise in hardware and low level programming languages a pre-requisite for customization. And the fragmentation of the environment – with different tools, practices and approaches – made it very difficult to run a combined lifecycle with common testing and deployment; often we could only test after a long and painful update on the IoT side.

For these reasons we decided to extend our vision of composition out of the cloud and into the IoT, aiming to find a way of bringing the same benefits of fast delivery, continuous deployment and simple management into the whole hyperconnected environment (Figure 1).

III. CREATING A CONVERGED PLATFORM FOR HYPERCONNECTED SYSTEMS

In considering the goals for our extended platform research we therefore set ourselves four major goals:

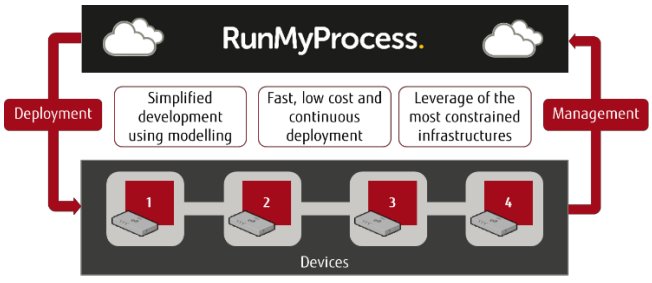

Figure 1- Bringing the best of web and cloud to the IoT via fast cycles and efficient deployment

1) *Simplified development:* Non-specialised developers should be able to build, deploy and test IoT applications. Given that accessibility of skills is a major barrier to the growth of the IoT, preserving the simplicity and accessibility of an established, model-based approach was a key goal in extending the scope of our platform to encompass the direct deployment of IoT device behavior.

2) *Flexible deployment:* Logic should be able to be deployed seamlessly to both the cloud and the IoT. We believe that we must simplify users' view of the world, enabling them to focus on the business logic they require without worrying about the wide range of nodes, topologies, protocols and other technical challenges that exist at the intersection of cloud and IoT.

3) *Easy integration:* When modelling functionality users should be able to easily connect resources whether they physically exist within the cloud or the IoT. We want to reduce unnecessary friction between cloud and IoT systems in order to simplify development of complex distributed systems and encourage innovation.

4) *Power optimized:* The approach should not require expensive firmware updates and distribution. Working with cheap, low cost devices means limiting the power, energy and bandwidth burdens our approach places onto the IoT hardware.

IV. DESIGNING A CONVERGED ENVIRONMENT

To map the capabilities we needed to generate IoT solutions, we began by building a conceptual architecture of the environment we wished to realize. This led to our intended functional architecture, as described in Figure 2:

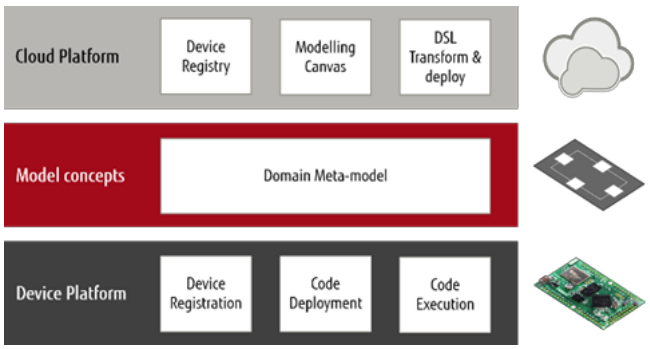

Figure 2 - Top level conceptual architecture

To meet our goals we had some particular requirements based on our prior experiences. Firstly we wanted to enable less skilled people to model the device functionality they needed and deploy it easily and immediately alongside the other cloud-based components that made up their overall solution. These components were assigned to our cloud platform layer. At the opposite end - to enable this - we also

wanted to eliminate the painful customization of firmware by relying on a device 'operating system' that could hide the differences in a broad range of standard devices and could be extended to offer an execution container for process definitions. These components were assigned to our device layer. But to make the two layers work together we needed to create a shared meta-model.

One of the most important elements of this architecture was therefore the domain meta-model of concepts needed to satisfy our implementation goals (Fig 3.).

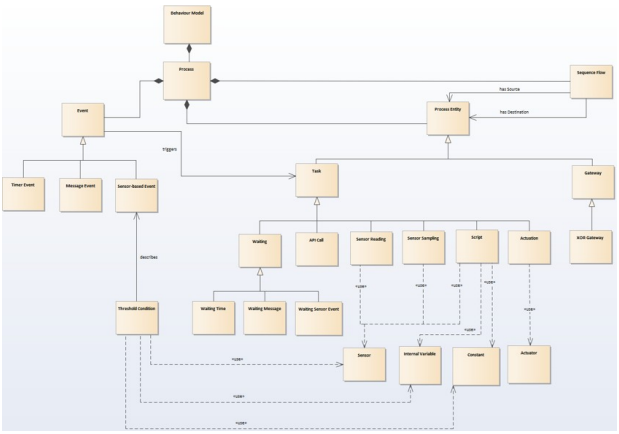

Figure 3 - Meta-model of solution space

By using a range of real world use cases provided by enterprises – spanning retail, utilities, facilities management and manufacturing - we were able to identify the core concepts that needed to be expressed within our models and supported by our execution environment – from steps within a process through to timers, sensor thresholds and external communication. In addition we used IPSO smart object definitions to capture the IoT specific information about the capabilities of each object. By building up and using this meta-model we were able to visualize requirements across the whole architecture, from the elements that needed to be modelled within our cloud environment to the capabilities our device runtime container needed to interpret and execute the modelled functionality (Figure 4).

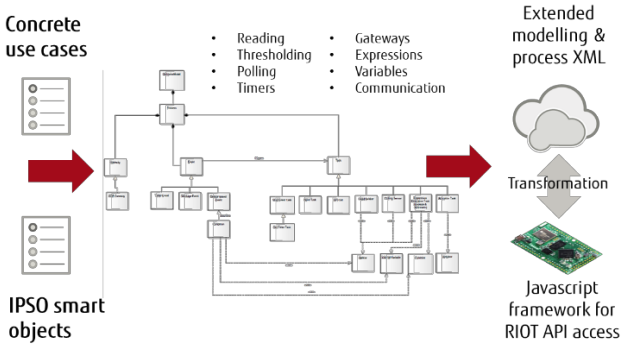

Figure 4 - Transforming inputs into components via a meta-model

V. PLATFORM IMPLEMENTATION

Together with our research partners we implemented the capabilities outlined in Figure 2, creating an environment in which is it possible to model and deploy functionality to the IoT in a simple, fast and iterative way (Figure 5)

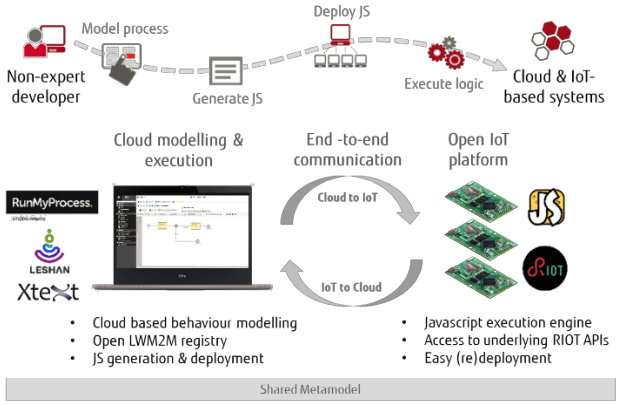

Figure 5 - End to end modelling and deployment for IoT

The implemented platform consists of three main elements – the cloud modelling and execution environment, the communication model and the open IoT platform. We will describe each in turn.

A. *Cloud modelling and execution*

The cloud modelling and execution environment is the main working environment for people using the platform. It provides the modelling, registry, deployment and management capabilities necessary for the creation of converged cloud-IoT solutions.

1) *Cloud based modelling:* Using RunMyProcess we built an IoT modelling canvas which enables users to graphically model and deploy functionality to be executed in the smart IoT node. It enables users to browse the available device types in the registry together with the configurations and available resources for each. We considered a range of different modelling techniques for building this functionality, in particular state machines, but after testing different approaches we concluded that BPMN 2.0 models provided the best match between the concepts we had dicovered in the meta-model and our ease of use requirements. In particular we appreciated the fact that (i) BPMN 2.0 is a language which reflects the way that non-IT users are accustomed to think and that (ii) it uses an XML-notation which was verbose enough to facilitate transformation into other assets such as code, increasing our architectural options. We also noted studies which suggested that BPMN is a more IoT-aware modelling approach when compared with alternatives such as eEPC (extended EPC) or UML activity diagrams [23]. A screenshot of the process modelled for the example scenario in section VI is presented in Figure 6.

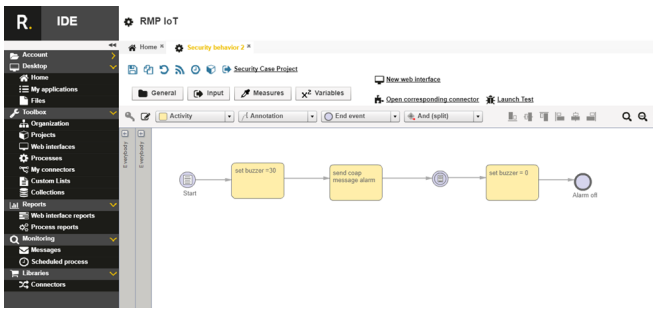

Figure 6 - IoT functionality modelled as BPMN

2) *Open LWM2M registry:* In order to keep track of the devices that we wish to manage and update we needed to integrate a repository into our existing cloud environment. Unlike with cloud deployment there is no central, homogenous runtime for the functionality being developed, so we need to understand the capabilities of each device and be able to choose which ones should receive any new behaviour. For IoT device management we chose to use the OMA Lighweight M2M standard [12], giving us a standards based model for interacting with devices. To implement this component we integrated Eclipse Leshan, an open source implementation of LWM2M written in Java [2]. It relies on Californium to provide CoAP and DTLS support and provides a simple web interface for viewing and interacting with the list of connected LWM2M clients. By integrating Leshan into our environment we have enabled people to choose the device type they wish to target new functionality towards.

3) *Javascript generation and deployment:* Once modelling is complete we need to deploy the functionality to the underlying devices. We considered a range of approaches for this but finally decided to transform the BPMN model into JavaScript, due to its widespread use and the availability of interpreters on a large number of device platforms. In order to achieve this the cloud environment dispatches the BPMN model to a domain specific language (DSL) server. Using the concepts from the meta-model the DSL server is able to transform the model into Javascript compatible with the device execution engine and its enabling APIs. We considered several language workbenches for the creation of the DSL server, including Xtext, MPS, spoofax, Rascal and ANTLR - an excellent comparison of the different features of these popular DSL workbenches can be found in [43] – but finally decided to adopt the eclipse-based Xtext. We made this decision due to its wide use across industry, ease of access through open source and the fact that it can be used to compile to any target language.

Once the BPMN model is transformed into Javascript it is then deployed to the chosen devices using CoAP.

B. *Cloud-IoT communication*

Over the last few years there has been significant and growing interest in the use of internet protocols to bridge the traditionally separate worlds of the Web and embedded devices in order to create an Internet of Things. One benefit of such an approach is the ability to enhance the discrete functionality of individual devices with additional logic in the cloud, using the speed and scale of cloud development to build more intelligent systems. While it has been demonstrated that existing Web architectures and protocols can successfully be used for the integration of constrained devices [36], it is unlikely that such an approach will be suitable for the types of limited capability devices we intended to support in this project.

For this reason we adopted the Constrained Application Protocol (CoAP) [28]. CoAP attempts to create a balance between consistency on the one hand and recognition of the limited power, reliability and bandwidth available to constrained devices on the other. When combined with the wider efforts of the IETF in creating 6LoWPAN [37] we can clearly see the foundations emerging for a Web of Things [38]. Support for CoAP within RunMyProcess was the subject of previous work [31].

In particular while CoAP is not a compression of HTTP it does offer a consistent interaction model along with best practices for mapping [39]. Using CoAP, individual resources continue to be identified and addressed via Universal Resource Identifiers (URIs), represented using arbitrary formats (such as JSON or XML) and manipulated using the same methods as HTTP. This use of a familiar approach will reduce the complexity of managing, integrating and updating distributed devices from our cloud platform.

C. *Open IoT platform*

As already discussed the use of an IoT "operating system" was a key enabler to our overall requirements, shielding developers from the complexities and cost of low level development. A number of IoT operating systems are available for the type of hardware and scenarios we wanted to enable [40]. In this project we chose to use RIOT [15][9] as the basic layer of our IoT device platform, an open source operating system with a low memory footprint, high modularity, good interoperability and support for a wide variety of low-end IoT devices.

In order to meet the goals of our project, however, we needed to add significant new functionality to the RIOT environment, including registration, deployment and execution support for our modelled code.

1) *Device Registration:* As discussed IoT device registration is implemented using Eclipse Leshan, which provides a LWM2M server implementation written in Java [2]. To facilitate the cloud to device interactions, we have implemented a minimal LWM2M client within the RIOT

environment to register with the server. To expedite our research, the bootstrap process uses simple pre-shared keys. As an example of the registration data exchanged at bootstrap, the LWM2M client in RIOT registers the resources < 3/0 >, < 9/0 >, < 3303/0 > and < 3315/0 > for the example scenario discussed in section VI, which map respectively to the IoT device, the JavaScript container, the light sensor and the sound-level sensor, as per the OMA standard [26].

2) *Javascript code deployment:* The code deployment component receives Javascript files and safely installs them into the memory of the device. Deployment messages are sent using CoAP. This approach is used as an alternative to the distribution of binaries, allowing small scripts to be dynamically updated and executed. This is an attractive approach as a result of the emergence of small footprint interpreters such as MicroPython [41] and JerryScript [42]. In this project, we have built upon a small footprint interpreter (JerryScript, as described below), and dynamically load scripts to roll out and update business logic on low-end IoT devices.

3) *Javascript code execution:* In order to support the execution of the modelled processes, the RIOT OS has been extended with a framework that interfaces between the Javascript logic and the underlying OS. This isolates the business logic from the underlying hardware and low level interfaces and provides a local execution environment for the generated Javascript. This environment enables e.g.:

- interaction with sensors and actuators;
- receipt of reliable, pre-processed data and events extracted from sensors, and
- remote interaction with the cloud or other devices in order to exhange information and trigger wider application components.

For the script engine, we chose to create a RIOT package for JerryScript [4], a lightweight JavaScript interpreter, and combined it with our new framework. This framework maps the low level APIs offered by RIOT – e.g. timers, sensor interactions, event callbacks etc – into more developer-friendly Javascript. Examples of the resulting RIOT JavaScript APIs are shown in Listing 1.

```
//Sensor & actuator access API
    sensor = saul.get_by_name("NAME");
    sensor = saul.get_one(TYPE);
//Sensor & actuator manipulation API
   sensor.on_threshold(LEVEL, callback, FLANK); sensor.read();
   actuator.write(VALUE);
//Network access API
   coap.register_handler(resource_name, COAP_METHOD, callback);
   coap.request(url, COAP_METHOD, payload);
// Timer API & snippets
   t = timer.setInterval(callback,interval_length_in_usec);
   t = timer.setTimeout(callback, timeout_in_usec);
```

Listing 1 - Example RIOT Javascript APIs

VI. TESTING

In order to test our work we chose a facilities management use case from the list of scenarios we used as input for our analysis work. This scenario is based on building security management and deals with intrusion detection and resolution. It uses an on-device process to detect and process a combination of light and sound information before making a local decision whether to trigger the attached alarm and notify a security guard via a cloud-based mobile notification (Figure 7).

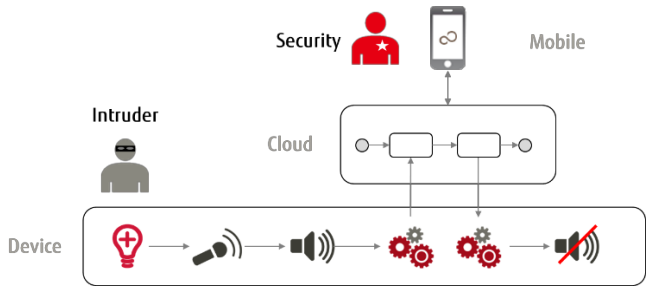

Figure 7 - Test scenario

Note that we chose this use-case only for practical, proof-of-concept purposes. The approach we present is generic and can easily be applied to a wide variety of other IoT use cases.

A. *Executing the scenario*

During the execution of the scenario, the IoT device monitors light levels looking for anomalies, with raised light levels set as the 'triggering event' within the process model. At deployment time a threshold event is created within the operating system by the Javascript. If an anomaly is detected then an event is sent from the operating system to the Javascript to trigger the rest of the code representing the modelled process (as shown in Figure 8).

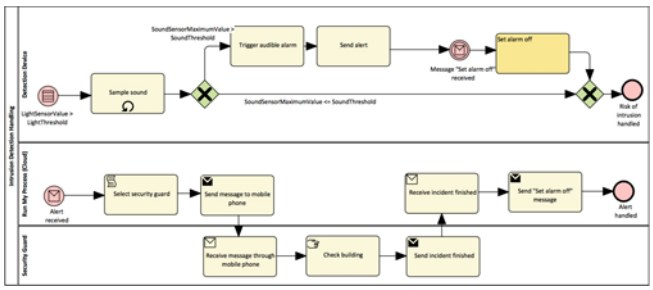

Figure 8 - Modelled security process (top layer is device process)

Once triggered, the code first listens for sound using the attached sensor in order to attempt to confirm that an intrusion is likely and reduce false alarms. An important point to note is that the implementation of the Javascript APIs used to interface with the underlying operating system pre-process the returned data to minimize noise and anomalies, reducing the need to deal with potential hardware issues within the business logic.

If sound is also detected than the alarm is triggered, a message is sent via CoAP to the cloud in order to trigger the security process and the device process enters a wait state.

On receiving the CoAP message the cloud platform starts a second process to manage attendance by the security guard. This process extracts the necessary data about location and issue and sends a notification via a mobile application implemented with RunMyProcess to alert the security guard. Once accepted the security guard can choose to cancel the alarm from their mobile device (see Figure 9).

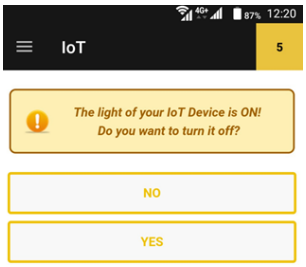

Figure 9 - Option to cancel the alarm on the mobile device

Once cancelled a message is sent back to the cloud platform, which in turn sends a CoAP message to the device in order to instruct it to switch off the alarm. On receipt of the event the device switches off the alarm, completes the process and returns to monitoring mode.

An overview of the end-to-end components involved in this process can be seen in Figure 10.

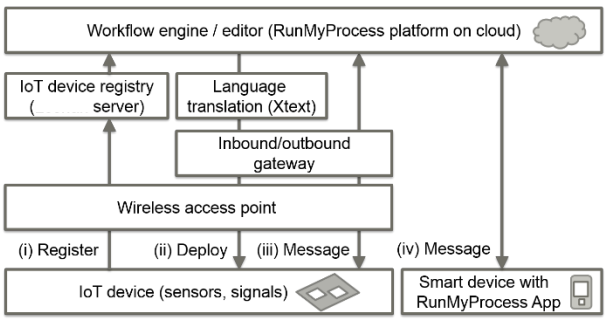

Figure 10 - Components used involved in implementation

Listing 2 illustrates the JavaScript implementation of the process model shown in Figure 8, demonstrating what is deployed to the device. As you will see this Javascript is able to interact with the underlying sensors, actuators and networking components in order to implement the behavior necessary to support the desired scenario.

```
var process_1 = new function() {

  this.brightness_1 = saul.get_by_name("brightness ");
  this.sound_1 = saul.get_by_name("sound");
  this.buzzer_1 = saul.get_by_name("buzzer");
  this.sound_level; var self =  this;
  this.start = function () {

    self.brightness_1.on_threshold(800.0, self.activity2);
  };

  this.activity2 = function () {
    self.sound_level = self.sound_1.sample(5000);
    self.split_xor3();
  };

  this.split_xor3 = function () {
    if (self.sound_level.max > 100.0) {
         self.activity5();
    } else {
         self.activity4();
    }
  };

  this.activity4 = function () {
    print('Light but no sound');
  };

  this.activity5 = function () {
    self.buzzer_1.write(100.0); self.activity7();
  };

  this.await7 = function () {
    var handler;
    var callback = function () {
      handler.cancel(); self.activity8();
    }

    handler = coap.register_handler("/alarm",
      coap.method.PUT, callback);
    coap.request("coap://[2a05:d014:677:XXXX:
YYYY:f713:6820:e17f]/coap", coap.method.POST, "ALARM!");
  };

  this.activity8 = function () {
    self.buzzer_1.write(0.0);
    print('Canceling alarm');
  };
 }

 process_1.start();
```

B. Hardware and connectivity

As many of our goals relate to the use of low power, low capability hardware we wanted to ensure that our approach worked for simple devices. In this section we will briefly outline the hardware and connectivity set up used for the IoT aspects of the project.

1) *IoT hardware:* In order to meet the requirements of the implemented scenario we produced a low-end IoT device. This device was based on commercially available equipment from Microchip, the SAMR21-xpro [5]. The SMAR21 features a 32- bit micro-controller, 32kBytes of RAM and 256 kBytes of Flash memory, together with an IEEE 802.15.4 radio transceiver. Note that the SAMR21 has no memory protection unit (MPU) in hardware. We extended the SAMR21 with a custom break-out board shown in Figure 11, connecting a light sensor, sound level sensor and revolving light via GPIO.

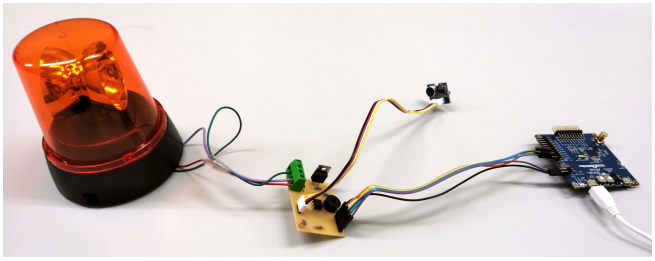

Figure 11 – IoT test hardware

2) *Connectivity:* The low-end IoT device was connected the to the Internet via a cheap RaspberryPi with a IEEE 802.15.4 radio module, configured in a standard fashion to act as a border router between plain IPv6 and the LoWPAN [22]. The setup is shown in Figure 12.

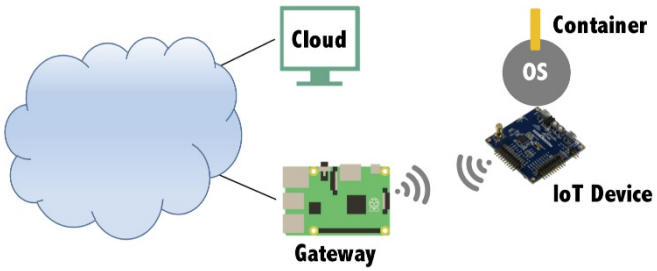

Figure 12 - Connectivity setup

VII. EVALUATION

At the outset of our project we identified four major goals, simplified development, flexible deployment, easy integration and power optimization. In this section we will briefly review our work in the context of these goals.

A. Simplified development

Our goal here was twofold – to enable non-specialised IoT developers to build IoT application and to enable a faster and more iterative approach to IoT development.

It is outside the scope of this paper to consider whether BPMN is a simple approach in general, but on the assumption that it provides a non-IoT specific way to model business functionality we believe this criteria has been met. From informal testing with colleagues and customers with no specific IoT development experience or skills we have observed that the extended cloud modeling environment enables non-specialized developers to build and deploy simple IoT applications. By using the adapted BPMN modelling language in combination with drag and drop, such users can graphically design the business logic they wish to be executed on the IoT devices.

Equally importantly the deployment of this functionality now happens at the click of a button, enabling a much more iterative cycle of modelling, deployment and testing. In fact we observed a side effect of this process during the implementation of our own test scenario. Because deployment can now be a continuous process rather than a single event we were able to quickly deploy the solution and tune thresholds and other parameters based on observation in situ rather than estimations. We believe that this experience could point to a significant additional benefit of our approach, enabling the faster, cheaper and lower risk deployment of IoT solutions due to the ease of adapting their behavior post-implementation.

B. Flexible deployment

Our extended cloud development environment offers the opportunity to model both cloud and IoT processes within the same environment and using the same concepts and models. Today the developer has to choose from the outset whether they wish the process to be targeted at the cloud or the IoT, since there are some additional concepts and deployment steps necessary for the IoT. As a result while we have succeeded in our main goal of enabling the deployment of processes to the cloud and the IoT *as part of the same project/solution* within the development environment, it is currently not possible to move such modelled processes *between* environments as a further optimization. This will be the subject of future research.

C. Easy integration

As demonstrated in our test scenario, processes within the cloud and IoT are able to easily inter-operate at runtime using CoAP. This means that it is simple to create 'hyperconnected' solutions which consist of different components, deployed in different environments, but which are still able to form part of the same over-arching solution through integration.

Equally importantly, from a development time perspective the inclusion of a basic device registry makes it easy to discover, introspect and connect devices and their behavior within process flows. This is an equally important perspective in terms of integration. Today much of the work to do this is

still manual, e.g. in terms of viewing and copying over information from the registry into the process flow. A topic of future research will therefore be the automation of this kind of activity to enhance developer productivity and reduce risk of errors.

D. Power optimized

In considering the viability of our approach from a resource perspective we have to look at two perspectives: (i) the memory overhead of running the Javascript engine vs directly implementing functionality in C and (ii) the comparative overhead of updating scripts vs firmware updates.

In the first case, we estimate that the total overhead of running the Javascript engine based on our analysis is ~12k of RAM and ~96k of flash memory. While this may look like an overhead vs a native C implementation, the efficiency of the RIOT OS means that it would rarely take the overall capacity requirements outside the scope of the majority of standard off-the-shelf low power devices (e.g. 32k of RAM and 256K of flash memory). In fact our analysis suggests that our solution could be readily compiled and run on more than 84 different types of IoT devices, corresponding to more than 80% of all the IoT hardware supported by RIOT today.

In the second case, the number of bytes transmitted over-the-air to update the functionality of the device for our Security Scenario is 1kB (the size of the script shown in Listing 2). This script is actually relatively complex with multiple accesses of sensors and actuators, calculations and external communications. Compared to this, a full firmware update would require the transmission of an entire image, which in the case of RIOT would be ~60kB. This is a significant penalty in comparison and would represent a challenge to the kind of iterative processes we have outlined. We acknowledge that partial firmware updates could reduce this overhead but believe that the installation process would continue to be more complex and incur a greater overhead.

While our approach therefore introduces some overhead at the outset we believe it could significantly reduce cumulative overheads across the entire lifecycle of a device, extending its life as well as delivering the already outlined benefits of agility and convergence.

VIII. RELATED WORK

Building today's IoT solutions is a challenging task because developers need to have extensive technical knowledge spanning embedded systems, networking and protocols. The resulting low level development and technical work not only makes the creation of IoT applications slow, error-prone and cumbersome but also significantly restricts the pool of people who can engage with IoT for solving real-world problems. As a result of these challenges, many researchers have been considering how to accelerate the creation of IoT applications. As we can see by reviewing the landscape, however, the primary approach taken to date has focused on the creation of cloud-based functionality - as opposed to IoT-based - limiting their utility in situations where a more federated approach is required.

In our previous research we proposed a cloud-based architecture for composing value from the IoT [30]. In this model our cloud platform made it possible to easily create systems that orchestrated IoT data outputs with other kinds of web and on-premise resources. It was the limitations of this approach during real-world testing, however, that led to the research outlined in this paper.

In [44], life cycles for IoT devices, services and applications are discussed. It gives hence an academic framework under which different aspect of IoT development can be discussed but gives no practical implementation.

Spacebrew [11] is an open software toolkit which enables outputs and inputs to be connected, creating an implicit 'flow'. However, it is based on a publisher/subscriber which may be challenging for non-professional software developers. Equally importantly Spacebrew cannot run functionality directly on devices, relying instead on intermediate computers that share data from the devices to which they are connected (e.g. Arduino).

Paraimpu [7] [25] is another software system proposed for simplifying the creation of systems which connect physical objects, APIs and services, composing them together to create new applications. Again, however, the focus here is on connecting data from existing objects rather than adapting their behaviour through the deployment of business logic.

Actinium [20] presents a web-like scripting approach for composing low-end devices, providing a RESTful, cloud-based run-time container with dynamic installation, update, and removal of scripts. While the goal of Actinium is also to simplify the development of converged systems, its approach focused on cloud-based scripts which do not change the behaviour of the underlying devices themselves.

[48] Discusses an approach similar to ours that uses the BPMN to model behavior of wireless sensor networks. They compile the BPMN edge device behavior to the callas language and execute it on the callas virtual machine. In contrast to that, we compile the BPMN process to javascript. According to [49], the callas virtual machine is only available for SunSpot devices whereas we expect that javascript interpreters will be available for a large number of different devices.

The WoTkit Processor, bundled with the WoTkit platform [17][18], is another cloud-based tool for building applications that use IoT resources. As with the other options, however, it focuses on aggregating sensor data within the cloud before building cloud-based applications that leverage its value.

Finally NodeRed [6] is a web-based tool for creating flows that connect hardware devices, APIs and cloud services, with modelling offered via a browser-based flow editor. From the perspective of our research, NodeRED is the closest prior work to our proposed solution, offering the ability to model and deploy functionality to the edge of the network. Node

RED still requires that flows are deployed to either the cloud or to general purpose computing devices with a traditional operating system (such as the Raspberry Pi [8]). For this reason NodeRED did not satisfy our requirement for behaviour deployment to constrained devices. Furthermore, while partially addressing the issues of development simplification through a flow editor, NodeRED did not address the wider issues of end- to-end process integration and device lifecycle management that are a core part of our research.

Several other platforms [21] [19] [33] [27] have also been proposed to ease the development of IoT applications but in all cases they do not cover the major features we propose, including integration of device registry, workflow editing, wireless deployment and end-to-end orchestration.

Besides these technical papers that discuss the creation of solutions for the IoT-domain, there are also a couple of papers that discuss new and emerging business models related to IoT [45,46,47]. We have shortly touched these topic at the beginning of our paper but refer the interested reader to the mentioned papers that discuss these topics in more detail.

IX. CONCLUSION

In this paper, we have examined one possible route to simplifying the development of new hyperconnected systems that span both cloud and IoT infrastructures. To date the ease of development of IoT applications has lagged the cloud significantly, lacking easy support for established concepts such as modelling, agile delivery and continuous deployment, concepts which have transformed the development of more mainstream software.

The work described in this paper aimed to address this gap by building development, deployment and management support for IoT devices based on established cloud practices. To this purpose we have created a cloud-IoT platform composed of a set of components: a generic IoT device operating system (RIOT), an accompanying 'psuedo-container' for the execution of business logic on low-end devices and a cloud-based development tool for rapid business-logic modelling and deployment, We have presented our test implementation in order to illustrate the workings of this platform and to help examine the ways in which our work can enable our core aims of simplified development, flexible deployment, easy integration and power optimization.

REFERENCES

[1] CoAP,RFC 7252 Constrained Application Protocol. http://coap.technology/,.

[2] Eclipse Leshan. https://www.eclipse.org/leshan.

[3] Gartner Says 4.9 Billion Connected "Things" Will Be in Use in 2015. https://www.gartner.com/newsroom/id/2905717.

[4] JerryScript RIOT Package. https://github.com/RIOT- OS/RIOT/tree/master/pkg/jerryscript.

[5] Microchip Atmel ATSAMR21-xpro Board. http://www.atmel.com/tools/atsamr21-xpro.aspx.

[6] NodeRED. https://nodered.org.

[7] Paraimpu. http://www.paraimpu.com/,.

[8] Raspberry Pi - Teach, Learn, and Make with Raspberry Pi. https://www.raspberrypi.org/,.

[9] RIOT - The friendly Operating System for the Internet of Things. https://riot-os.org/.

[10] RunMyProcess. https://www.runmyprocess.com.

[11] Spacebrew. http://docs.spacebrew.cc.

[12] O. M. Alliance. Lightweight machine to machine technical specification. *Technical Specification OMA-TS-LightweightM2M-V1*, 2013.

[13] T. Allweyer. *BPMN 2.0: introduction to the standard for business process modeling*. BoD–Books on Demand, 2016.

[14] E. Baccelli et al. Scripting Over-The-Air: Towards Containers on Low- end Devices in the Internet of Things. In *IEEE Percom, 2018*.

[15] E. Baccelli et al. RIOT OS: Towards an OS for the Internet of Things. In *32nd IEEE INFOCOM. Poster*, Turin, Italy, 2013. IEEE.

[16] P. Biggs, T. Johnson, Y. Lozanova, and N. Sundberg. Emerging issues for our hyperconnected world. *The global information technology report*, pages 47–56, 2012.

[17] M. Blackstock and R. Lea. Iot mashups with the wotkit. In *Internet of Things (IOT), 2012 3rd International Conference on the*, pages 159–166. IEEE, 2012.

[18] M. Blackstock and R. Lea. Wotkit: a lightweight toolkit for the web of things. In *Proceedings of the Third International Workshop on the Web of Things*, page 3. ACM, 2012.

[19] N. K. Giang, M. Blackstock, R. Lea, and V.C. Leung. Developing iot applications in the fog: a distributed dataflow approach. In *Internet of Things (IOT), 2015 5th International Conference on the*, pages 155–162. IEEE, 2015.

[20] M. Kovatsch et al. Actinium: A restful runtime container for scriptable internet of things applications. In *IEEE IoT, 2012*.

[21] M. Kovatsch, M. Lanter, and Z. Shelby. Californium: Scalable cloud services for the internet of things with coap. In *Internet of Things (IOT), 2014 International Conference on the*, pages 1–6. IEEE, 2014.

[22] N. Kushalnagar, G. Montenegro, and C. Schumacher. Ipv6 over low- power wireless personal area networks (6lowpans): overview, assump- tions, problem statement, and goals. Technical report, 2007.

[23] S. Meyer, K. Sperner, C. Magerkurth, and J. Pasquier. Towards modeling real-world aware business processes. In *Proceedings of the Second International Workshop on Web of Things*, page 8. ACM, 2011.

[24] D. Miorandi, S. Sicari, F. De Pellegrini, and I. Chlamtac. Internet of things: Vision, applications and research challenges. *Ad hoc networks*, 10(7):1497–1516, 2012.

[25] A. Pintus, D. Carboni, and A. Piras. Paraimpu: a platform for a social web of things. In *Proceedings of the 21st International Conference on World Wide Web*, pages 401–404. ACM, 2012.

[26] S. Rao et al. Implementing LWM2M in constrained IoT devices. In *IEEE ICWiSe, 2015*.

[27] A. Salihbegovic, T. Eterovic, E. Kaljic, and S. Ribic. Design of a domain specific language and ide for internet of things applications. In *Information and Communication Technology, Electronics and Mi- croelectronics (MIPRO), 2015 38th International Convention on*, pages 996–1001. IEEE, 2015.

[28] Z. Shelby et al. RFC 7252: Constrained Application Protocol (CoAP). *IETF Request For Comments, 2014*.

[29] L. Tan and N. Wang. Future internet: The internet of things. In *Advanced Computer Theory and Engineering (ICACTE), 2010 3rd International Conference on*, volume 5, pages V5–376. IEEE, 2010.

[30] I. Thomas, S. Gaide, and M. Hug. Composite business ecosystems for the web of everything: using cloud platforms for web convergence. *In Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2013 Seventh International Conference on, pages 535–540. IEEE, 2013*

[31] I. Thomas, S. Ziegler, C. Crettaz, L. Fedon, and S. Gaide. Making

it all work together enabling new business models in the web of everything. *In Internet of Things (WF-IoT), 2014 IEEE World Forum on, pages 526–531. IEEE, 2014*

[32] O. Vermesan and P. Friess. Building the hyperconnected society: Internet of things research and innovation value chains, *ecosystems and markets, volume 43. River Publishers, 2015*

[33] R. Vilalta, A. Mayoral, D. Pubill, R. Casellas, R. Mart´ınez, J. Serra, C. Verikoukis, and R. Mun˜oz. End-to-end sdn orchestration of iot services using an sdn/nfv-enabled edge node. *In Optical Fiber Communication Conference, pages W2A–42. Optical Society of America, 2016.4*

[34] I. Thomas, L. Fedon, A. Jara, Y. Bocchi. Towards a Human Centric Intelligent Society: Using Cloud and the Web of Everything to Facilitate New Social Infrastructures. In 9th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing, 2015

[35] Transforming Financial Services for a hyperconnected world. Fujitsu Forum 2015. https://www.youtube.com/watch?v=URW557QWLhI

[36] D. Guinard, V. Trifa, and E. Wilde, A Resource Oriented Architecture for the Web of Things, Proc. IEEE Intl Conf. on the Internet of Things, Nov.2010.

[37] N. Kushalnagar, G. Montenegro, C. Schumacher, IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement & Goals, *IETF RFC 4919, Aug. 2007*

[38] Deze Zeng, Song Guo, and Zixue Cheng, The Web of Things: A Survey (Invited Paper), *Journal of Communications, vol. 6, Sept. 2011, pp. 424-438.*

[39] A. Castellani, S. Loreto, A. Rahman, T. Fossati, E. Dijk, Best Practices for HTTP-CoAP Mapping Implementation, *Internet Draft, draft-castellani-core-http-mapping-07*

[40] O. Hahm et al. Operating systems for low-end devices in the internet of things: A survey, *IEEE Internet of Things Journal, no. 5, pp 720-734, Oct. 2016*

[41] MicroPython Compiler and Runtime https://github.com/micropython/micropython

[42] JerryScript: JavaScript engine for the Internet of Things https://github.com/jerryscript-project/jerryscript

[43] Erdweg, Sebastian, et al. Evaluating and comparing language workbenches: Existing results and benchmarks for the future. *Computer Languages, Systems & Structures 44 (2015): 24-47*

[44] Leila Fatmasari Rahman, Tanir Ozcelebi, Johan Lukkien, Understanding IoT Systems: A Life Cycle Approach, Procedia Computer Science,Volume 130, 2018, pp 1057-1062

[45] Concetta Metallo, Rocco Agrifoglio, Francesco Schiavone, Jens Mueller, Understanding business model in the Internet of Things industry, Technological Forecasting and Social Change Journal ,2018, ISSN 0040-1625,

[46] Vlad Krotov, The Internet of Things and new business opportunities, Business Horizons,Volume 60, Issue 6, 2017, pp 831-841

[47] Jaehyeon Ju, Mi-Seon Kim, Jae-Hyeon Ahn,Prototyping Business Models for IoT Service, Procedia Computer Science,Volume 91, 2016, pp 882-890,

[48] Francisco Martins, Dulce Domingos, Modelling IoT behaviour within BPMN Business Processes, Procedia Computer Science, Volume 121, 2017, pp 1014-1022,

[49] Rui Mendes and Luis Lopes, A Modular Virtual Machine for the Callas Programming Language for Wireless Sensor Networks, Simposia Informatica (INForum 2011), Coimbra, Portucal

[50] Thomas, Ian et al. Composite Business Ecosystems for the Web of Everything: Using Cloud Platforms for Web Convergence, 2013 Seventh International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (2013): 535-540.